UNIVERZA V MARIBORU
FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO

Marko Dolar

ANIMACIJA OGNJA S SISTEMI DELCEV

Diplomsko delo

Maribor, september 2014

ANIMACIJA OGNJA S SISTEMI DELCEV

Diplomsko delo

Študent:	Marko Dolar
Študijski program:	Visokošolski študijski program
	Računalništvo in informacijske tehnologije
Mentor:	red. prof. dr. Nikola Guid
Somentor:	viš. pred. dr. Simon Kolmanič

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

Številka: E1021015
Datum in kraj: 08. 04. 2014, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 46/2012)
izdajam

SKLEP O DIPLOMSKEM DELU

1. **Marku Dolarju,** študentu visokošolskega strokovnega študijskega programa RAČUNALNIŠTVO IN INFORMACIJSKE TEHNOLOGIJE, se dovoljuje izdelati diplomsko delo pri predmetu Računalniška grafika in animacija.

2. **MENTOR:** **red. prof. dr. Nikola Guid**
   SOMENTOR: **viš. pred. dr. Simon Kolmanič**

3. **Naslov diplomskega dela:**
   ANIMACIJA OGNJA S SISTEMI DELCEV

4. **Naslov diplomskega dela v angleškem jeziku:**
   ANIMATION OF FIRE USING PARTICLE SYSTEMS

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije do 30. 09. 2014 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:
red. prof. dr. Borut Žalik

Obvestiti:
- kandidata,
- mentorja,
- somentorja,
- odložiti v arhiv.

ZAHVALA

Zahvaljujem se red. prof. dr. Nikoli Guidu in viš. pred. dr. Simonu Kolmaniču za strokovni pregled, pomoč in nasvete pri pripravi diplomskega dela.

Animacija ognja s sistemi delcev

Ključne besede: računalniška grafika, sistemi delcev, animacija, ogenj

UDK: 004.92(043.2)

Povzetek

Sistem delcev je na področju računalniške grafike modelirna tehnika za postopkovno gradnjo dinamičnih objektov. Najpogosteje se uporablja pri vizualizaciji naravnih pojavov, kot so sneg, padajoči slap in oblaki. Te sestavimo iz velikega števila manjših delcev, ki so si med seboj podobni. Končni izdelek diplomskega dela je animacija ognja, ki se lahko izrisuje interaktivno. Pri delu smo preizkušali različne načine za upodobitev sistema in animacijo posameznih delcev.

Animation of Fire Using Particle Systems

Key words: computer graphics, particle systems, animation, fire

UDK: 004.92(043.2)

Abstract

A particle system is a modeling technique for procedural building of dynamic objects in the field of Computer Graphics. It is most used for visualization of natural phenomena, like snow, waterfall, or clouds. They consist of many small particles that are similar among each other. The final product of this diploma is an animation of fire that can be drawn interactively. While working, several techniques were tested for rendering a system and animation of individual particles.

Kazalo vsebine

1. Uvod....1
2. Sistemi delcev....3
2.1. Splošen pregled zgradbe in delovanja....3
2.2. Zgodovinski mejniki....4
2.3. Prednosti....7
3. Opis delovanja....9
3.1. Ustvarjanje delcev....10
3.2. Lastnosti delcev....11
3.3. Dinamičnost....12
3.4. Ponor delcev....15
3.5. Izris....15
3.6. Upravljanje s sistemi....16
4. Praktične izvedbe....18
4.1. Obstoječe rešitve....18
4.2. Primeri življenjskih ciklov....19
5. Animacija ognja....23
5.1. Programski vmesnik OpenGL....23
5.2. Kamera....24
5.3. Nadzor nad sistemi....26
5.4. Shramba delcev....26
5.5. Izvor delcev....27
5.6. Lastnosti delcev....29
5.7. Gibanje delcev....30
5.8. Zaznavanje trka in odboj....33
5.9. Izris delcev z OpenGL....35
5.10. Uporabniški vmesnik....38
5.11. Pohitritev izvajanja....40
6. Sklep....41
7. Seznam uporabljenih virov....42

Kazalo slik

Slika 2.1: Pretvorba planeta iz filma »The Wrath of Khan«.....5
Slika 2.2: Uvodni del animacije »The Adventures of André and Wally B.«.....6
Slika 3.1: Hierarhija sistemov delcev.....17
Slika 4.1: Sistem delcev v 3D modelirniku Blender2.71.....19
Slika 4.2: Eksplozija delcev ter teksture uporabljene pri izrisu [Wikipedia, 2014].....21
Slika 4.3: Uporaba delcev pri izrisu travnih bilk [Reeves, 1983].....22
Slika 4.4: Spletna interaktivna galaksija stars.chromeexperiments.com [1. 9. 2014].....22
Slika 5.1: Primer začetnega položaja delca v krogu z nespremenljivim vektorjem hitrosti 29
Slika 5.2: Lastnosti delca – položaj in smer ter velikost amplitude in frekvence nihanja…30
Slika 5.3: Preslikava vektorja skozi vektor normale.....34
Slika 5.4: Izris delca s prehodom.....37
Slika 5.5: Izris ognja, isker in dima.....38
Slika 5.6: Primer urejanja dveh izvorov delcev, ki ustvarjata ogenj.....39

Kazalo tabel

Tabela 5.1: Čas računanja novih položajev delcev s pohitritvijo.....40

Kazalo izpisov

Izpis 5.1: Izračun transformacijske matrike….....25
Izpis 5.2: Posodobitev delcev.....….....31
Izpis 5.3: GPU program, ki izračunava položaje delcev.....36
Izpis 5.4: GPU program, ki obarva delce s prehodom.....37

1. Uvod

Ljudje že vse od pojava grafičnih vmesnikov poskušamo čim bolj realistično simulirati naravo in predmete v navideznem okolju. Najbolj razširjena metoda za modeliranje objektov v računalniški grafiki temelji na ideji, da je vsak objekt sestavljen iz mnogokotnikov, ki si delijo nekatere robove in oglišča. Z njimi lahko gradimo trdne objekte (*solid objects*) kot so zgradbe in pokrajine, s pomočjo raznovrstnih animacijskih tehnik pa tudi živa bitja in njihovo gibanje. S hitrim razvojem računalniške industrije in širitvijo njene uporabe na druga področja se je pojavila potreba po realističnem in učinkovitem izrisovanju dinamičnih objektov, s katerimi bi lahko vizualizirali raznovrstne naravne in umetne pojave, kot so tekoča voda, eksplozije, sneženje pa tudi galaksije. Sčasoma se je rodil pojem sistemi delcev (*particle systems*), ki jih je kot prvi opisal William Reeves v svoji publikaciji [Reeves, 1983]. S pomočjo te metode je lažje ustvariti številne dinamične prostorske objekte, ki nimajo neke stalno določene oblike, saj se objekt neprestano giba in spreminja v svojem začrtanem območju.

Takšen način se je izkazal uporaben tudi pri modeliranju in animaciji nekaterih drugih objektov, kot so tekstil in lasje. Tehnika je nepogrešljiva tudi pri ustvarjanju scen z velikim številom neodvisnih a podobnih objektov, kot na primer številna drevesa v gozdu, jata ptic ali simulacija množice ljudi. Ročno ustvarjanje takšnih scen je sicer časovno zelo potratno delo.

Ker je človeške oči lahko ukaniti, ogenj ni zelo težko poustvariti z računalniško grafiko, to pa zaradi njegovega značilnega žarečega oranžno-rumenega videza. A kakor vidimo že iz prve publikacije pa animacija takšnega sistema sodi v ene izmed zahtevnejših. Sčasoma so se pri upodobitvi ognja uveljavili sistemi, kot je hidrodinamika (*fluid dynamics*), ki modelirajo ogenj precej podobno naravi, vendar so ti postopki precej zahtevni, saj vsebujejo procesorsko zelo zahtevne izračune. Ti modeli so lahko uporabljeni v filmski industriji, saj se lahko posamezne slike, na katerih se izvaja vizualni učinek, upodabljajo

dalj časa. Za izrisovanje interaktivne računalniške grafike (kot npr. v igrah) pa moramo poenostaviti izračunavanje položajev delcev, tako da dobimo rezultat, ki se lahko izvaja dovolj hitro na današnjih osebnih računalnikih. Pri tem objekt animiramo s pomočjo preprostejših fizikalnih zakonov, kot so pospešek, vrtenje in odboj.

Diplomsko delo je razdeljeno na 5 poglavij. Po trenutnem uvodnem poglavju na splošno opišemo sisteme delcev, ugotovimo katere prednosti imajo in kako so se razvili skozi čas. V tretjem poglavju so opisane osnove, na katerih gradi veliko upodobitev sistemov delcev. To poglavje vključuje ustvarjanje delcev, njihovo dinamičnost in končni izris množice. V sledečem, četrtem poglavju, se približamo praktičnim izvedbam z naštetimi primeri različnih sistemov in opisanimi obstoječimi programskimi orodji za delo z delci. Peto poglavje je namenjeno realizaciji specifičnega sistema v obliki animacije ognja. Tu je predstavljena in podrobneje opisana končna aplikacija. V zadnjem delu pa je diplomsko delo zaključeno s sklepom.

2. Sistemi delcev

Sistem delcev je v računalniški grafiki uporabna tehnika za modeliranje objektov, ki nimajo točno določene geometrične oblike. Sestavljen je iz številčne množice podobnih elementov, ki modelu določijo obliko in videz. Namen celotnega objekta je snovanje in upravljanje preprostih prostorskih teles z ustreznimi parametri, ki bodo kot celota dosegli želen učinek. S sistemom delcev lahko poživimo računalniško grafiko in jo naredimo bolj realistično, zato se ta tehnika veliko uporablja za prikaz različnih naravnih pojavov in ostalih vizualnih učinkov v računalniških igrah in produkcijskih filmih.

2.1. Splošen pregled zgradbe in delovanja

Dinamičnim objektom, kot so oblaki, ogenj in tekoča voda, se stalno spreminja oblika in izgled. Za opisane pojave je težko ustvariti 3D model, če ga želimo zgraditi iz mnogokotnikov, še težje pa ga je kot takšnega animirati. Ker imamo pri sistemih delcev kot osnovni gradnik posamezen delec, ga lahko enostavno nadziramo in mu spreminjamo njegove lastnosti, saj ponavadi ni odvisen od drugih elementov iz množice.

Gradniki enega objekta imajo podobne karakteristike. Vsak posamezni delec je predstavljen s točko v prostoru, ki pa ima lahko dodatne lastnosti, kot so barva, velikost in smer premikanja. Predstavljeni so lahko na različne načine, kot naprimer s štirikotnimi teksturami, preprostimi ali kompleksnimi geometričnimi oblikami ali pa samo z obarvanimi zaslonskimi pikami. Delci se lahko s časom premikajo ter spreminjajo svojo obliko, videz in/ali obnašanje. Lahko se konstantno premikajo, so za stalno prilepljeni na neko površino ali pa izginejo kmalu po nastanku.

Sistem delcev je največkrat uporabljen pri vizualizaciji različnih naravnih pojavov. Delci pri teh objektih imajo podoben življenjski cikel. Ti nastajajo v izvoru efekta, odigrajo svoj scenarij, kmalu zatem pa izginejo iz sistema. Tehnika se lahko uporabi tudi pri modeliranju

drugih nestandardnih oblik objektov, kjer obstaja veliko posameznih elementov s podobnimi karakteristikami. Z njimi si lahko pomagamo tudi pri izrisu vlaken, kot so lasje in puh. V takšnih sistemih se delci ustvarijo samo enkrat, na začetku. Skozi čas ne premikajo svojega izvornega položaja, lahko pa animiramo posamezna vlakna. Takšno obnašanje ima tudi trava. Sistem lahko uporabimo tudi pri animaciji množice z velikim številom ljudi ali drugih entitet, kjer se posamezniki premikajo v nekem prostoru in se izogibajo ostalim entitetam. Tehnika je uporabljena tudi pri vizualizaciji in animaciji tekstila, kjer so posamezni delci povezani med seboj v mrežo, razpon dolžine teh povezav pa določajo razteznost oblačila.

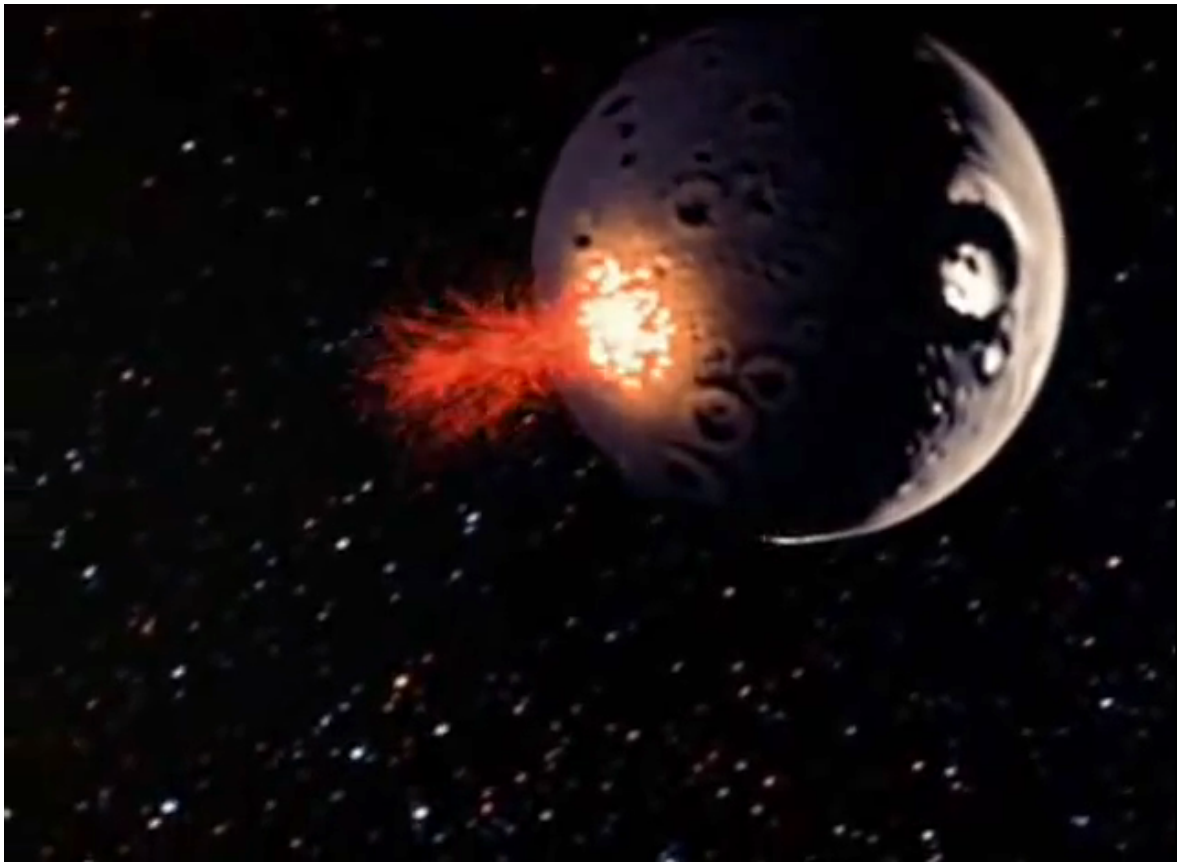
2.2. Zgodovinski mejniki

Sisteme delcev so nevedoč prikazovale že zgodnje računalniške video igre. Leta 1962 je izšla ena od prvih digitalnih računalniških iger »Spacewar!«. Ustvarjena je bila v zasebni raziskovalni univerzi MIT[1]. To je igra za dva igralca, kjer vsak nadzoruje svojo vesoljsko ladjo in poskuša zadeti nasprotnika. Ob zadetku nasprotnega igralca njegova ladja eksplodira v obliki oblaka naključno premikajočih se belo obarvanih zaslonskih pik. Igra »Asteroids« iz leta 1979 pa je bila verjetno prva igra, ki je v računalniški grafiki simulirala fizikalna pravila. Vsa grafika za igro je bila določena in izrisana v vektorski obliki. Pri tej arkadni strelski igri je igralec z izstrelki razstreljeval asteroide na manjše delce. Iz večjih gmot so ob zadetku nastajali manjši, a hitrejši asteroidi.

Ta tehnika se je že včasih veliko uporabljala pri vizualizaciji astronomskih teles za televizijske in znanstvene namene. V znanstveni dokumentarni seriji »Cosmos« sta Alvy Ray Smith in Jim Blinn uporabila delce za vizualne učinke, kot sta stvarjanje in izumiranje zvezd v galaksijah. Blinn je v eni od svojih publikacij [Blinn, 1982] opisoval upodobitev odboja svetlobe, ki prehaja čez sloj delcev. V njej sicer ni raziskoval ustvarjanja in zgradbe teh dinamičnih objektov, vendar so njegovo tehniko nadaljnje uporabili pri vizualizaciji slik Saturnovih obročev, ki pa so seveda sestavljeni iz majhnih prašnih delcev.

1 Massachusetts Institute of Technology

Sistem delcev je prvič omenil in definiral William T. Reeves v svoji publikaciji [Reeves, 1983]. Avtor je ob času pisanja delal kot animator vizualnih učinkov v filmskem in televizijskem produkcijskem podjetju Lucasfilm. V svoji publikaciji je predstavil tehniko ter definiral osnovni model in življenjski cikel delcev. Opisan koncept se ni veliko spreminjal in je do danes skoraj isti. V besedilu je tudi podrobneje opisana uporaba predstavljene metode pri upodobitvi transformacije planeta s pomočjo zidu ognja in eksplozij, kar so takrat uporabili kot vizualni učinek v filmu »Star Trek II: The Wrath of Khan« leta 1982. Izsek končnega efekta je viden na sliki 2.1.



Slika 2.1: Pretvorba planeta iz filma »The Wrath of Khan«

V kratkem risanem filmu »The Adventures of André and Wally B.« so tehniko sistema delcev uporabili pri izrisu trave na gozdnih tleh ter postavitvi številnih dreves po pokrajini, kar je razvidno iz slike 2.2. Pionirska animacija je z izidom leta 1984 v filmski industriji požela veliko zanimanja za računalniško animacijo – poleg prve uporabe zamegljevanja gibanja (*motion blur*) tudi zaradi kompleksnega osvetljenega gozdnega ozadja.

Slika 2.2: Uvodni del animacije »The Adventures of André and Wally B.«

Metoda se veliko uporablja v današnjih interaktivnih računalniških igrah, pri katerih se morajo objekti posodobiti in izrisati večkrat na sekundo. Ena prvih znanih in vplivnih iger, ki je uporabljala delce, je bila Quake iz leta 1996. Ta je izrisovala iskre, eksplozije in ostale vizualne učinke, ki jih pogosto najdemo v strelskih igrah.

S pojavom grafičnih procesnih enot so se lahko objekti s pomočjo strojnega pospeševanja izrisovali hitreje. Ti čipi vsebujejo mnogo paralelnih procesorjev, kar je idealno za premikanje in izris številčnih delcev. Po izidu programljivih grafičnih kartic, ki so podpirale grafična programska vmesnika Direct3D 8 in OpenGL 2.0, se je preračunavanje delcev lahko delno začelo seliti v GPU. Izračun položajev in drugih lastnosti delcev so se lahko izračunali na različne načine, čeprav ni bilo v grafičnem cevovodu (*graphical pipeline*) namenskih ustreznih zmogljivosti za njihov izračun. Z nedavno dodanimi GPU programi za preračunavanje (*compute shaders*) v Direct3D 11 in OpenGL 4.3, se lahko sistemi delcev izvajajo v celoti na grafični strojni opremi in tako razbremenijo CPU in prenos podatkov med enotama, ki je pogosto ozko grlo izvajanja. S pomočjo teh grafičnih pospeševalnikov lahko danes interaktivno izrisujemo na tisoče posameznih delcev, posledično pa končni rezultati izgledajo vse bolj osupljivi in najpomembneje realistični.

2.3. Prednosti

Ko modeliramo prostorske objekte s spreminjajočo obliko, ima pristop s sistemi delcev nekaj pomembnih prednosti napram uveljavljenemu grajenju objektov z mnogokotniki.

Osnovni gradnik vsakega sistema si lahko predstavljamo kot točko, ki je umeščena v nek prostor. Točka je najosnovnejši geometrijski objekt, ki nima nobene razsežnosti. Zaradi preprostosti tega objekta je obdelava njegovih podatkov veliko hitrejša, kakor pri mnogokotnikih. Slednji so namreč sestavljeni iz najmanj treh točk, imajo pa tudi dodatne lastnosti, kot so površina in robovi. Ti posledično naredijo lik bolj zapleten ter počasnejši pri izračunu parametrov za končen izris njihovih površin.

Objekt vsebuje mnogo delcev. Vsak posameznik ima naključno nastavljene začetne vrednosti, ki pa so vseeno določene v nekem začrtanem območju. Število delcev, ki jih lahko ustvarimo, je poljubno. Z nastavljanjem njihove količine z lahkoto spreminjamo gostoto objekta, saj je objekt narejen postopkovno in voden z naključnimi števili. Za modeliranje modela iz mnogokotnikov z veliko podrobnosti ponavadi porabimo veliko več časa. Velikokrat moramo za različne stopnje podrobnosti (*level of detail - LOD*) tudi ustvariti druge objekte ali jih preoblikovati in jih pri izrisovanju ob različni oddaljenosti

izmenjavati. Z delci lahko hitro ustvarimo in nadziramo ter uravnavamo stopnjo podrobnosti objekta za najboljši končni izgled. S to prednostjo lahko pri oddaljevanju objekta prikažemo manj delcev in tako razbremenimo procesno enoto pri računanju podatkov za končni izris.

Dinamični objekti, ki so zgrajeni s sistemom animiranih delcev, izgledajo, kakor da bi bili živi, saj svoj videz in/ali obliko spreminjajo vsak slikovni okvir. Pri navadnih ploskovnih modelirnih tehnikah je animacija težje dosegljiva, saj so osnovni gradniki zlepljeni med seboj.

3. Opis delovanja

Sistem delcev je ogromna množica majhnih delcev, ki kot celota predstavljajo en sam posamezen objekt. Vsak delec objekta je najprej ustvarjen, skozi čas se spreminja in premika po sistemu, ob nekem dogodku pa je iz sistema odstranjen.

Da animiramo sistem delcev, ga moramo izrisati večkrat na sekundo. Postopek izvedbe, ki se ponavlja za vsako izrisano sliko, lahko razdelimo na dva dela. V prvem simulacijskem načinu posodobimo vrednosti vsakega posameznega delca, tako da se ujemajo s trenutnim stanjem preostalega okolja. Tu lahko delce tudi ustvarjamo in jih brišemo iz množice. Ko imajo delci osvežene svoje lastnosti, lahko celoten objekt izrišemo, kar pa spada k drugemu delu izvedbe. Če prvo stopnjo razdelimo na posamezne dele in združimo z drugo, dobimo življenjski proces, ki se izvede posamezno za vsako izrisano sliko. Veliko izvedb sledi procesu, ki ga je prvič omenil že Reeves:

1. Tvorimo nove delce in jih dodamo v sistem.
2. Vsakemu novemu delcu posamezno priredimo svojevrstne lastnosti.
3. Vsi delci, ki so prekoračili svojo življenjsko dobo, so izbrisani iz množice.
4. Preostali delci se glede na svoje dinamične vrednosti preoblikujejo in svoj položaj premaknejo naprej glede na pretekli čas od prejšnjega do trenutnega izrisa.
5. Vsi delci se izrišejo na končno sliko.

Ta cikel je samo splošen primer, a je eden najbolj razširjenih. Sistem delcev je namreč lahko izveden kakorkoli, lahko izvede kakršnakoli navodila ob danem koraku. Ker je ta pristop postopkoven, lahko vsebuje kakršen koli računski model, ki opisuje izgled in obnašanje sistema. Na kar pa moramo paziti pri interaktivni grafiki, je hitrost izvajanja sistema pri posodobitvi vseh vrednosti in izrisu celote.

3.1. Ustvarjanje delcev

Delci so ustvarjeni v sistem naključno, a vendar na neko kontrolirano, smiselno in omejeno lokacijo. En delec lahko tudi tvori več drugih delcev.

Območje v okolju, kjer so novi delci ustvarjeni s svojimi začetni vrednostmi, imenujemo izvor (*emitter*). Ponavadi ima neko geometrično obliko, med katerimi pogosto srečamo točko, črto, kroglo ali pravokotnik. Oblika izvora sicer ni vidna na končnem izrisu, se pa nekatere začetne vrednosti delcev (kot so položaj in smerni vektor) ponavadi določijo glede na njegovo obliko. Delci se ustvarijo naključno v območju tega telesa, lahko v njegovi notranjosti (ko je izvor tri-dimenzionalen), na njegovi površini, robovih objekta ali pa v posameznih ogliščih.

Vsak slikovni okvir (*frame*) ustvarimo nekaj novih delcev. S količino novih elementov v sistemu vplivamo na gostoto izrisanega objekta. Pri eni od osnovnih metod lahko nadziramo povprečje, koliko delcev se bo ustvarilo pred vsakim novim izrisom slike. Za več naključnosti v sistemu še dodamo spremenljivko, ki določi največje možno odstopanje v številu novih delcev. Tako dobimo enačbo (3.1), ki nam določi število novih delcev, ki jih ustvarimo v trenutku.

$$n = n_a + r\, n_v \tag{3.1}$$

Tu je:
n – število novih delcev,
n_a – povprečno število novih delcev,
r – naključno število v območju [-1, 1] in
n_v – največje možno odstopanje števila delcev oz. variabilnost.

Izračun števila novih delcev lahko še popravimo tako, da je število novih delcev odvisno od oddaljenosti objekta od kamere. To pomeni, da manj prostora, ki ga objekt zajema ob končnem izrisu, manj delcev vsebuje. S takšno spremenljivko v bistvu nadziramo stopnjo podrobnosti modela. S tem lahko močno razbremenimo procesorski čas računanja, ki je potreben za posodobitev tisočih delcev. To se zgodi v primeru, ko izrisujemo objekt, ki

gledan od blizu vsebuje mnogo delcev, a je v naši sceni precej oddaljen, tako da ga vidimo samo čez nekaj zaslonskih pik. Pri tem izračunu rezultat iz enačbe (3.1) zmnožimo s skalarno vrednostjo *s*, ki predstavlja povečavo/zmanjšanje vidnega območja sistema glede na njegovo običajno velikost.

Prejšnja izračuna delujeta pravilno samo v primeru, če je čas med posameznimi okvirji konstanten. To je možno v primeru filmov ali v drugih animacijah, ki so upodobljena pred gledanjem končnega rezultata. Če želimo sistem risati interaktivno v trenutnem času, kot na primer v igrah ali simulacijah, potem moramo upoštevati različne časovne razmake med slikovnimi okvirji. Seveda lahko določimo njihovo število, vendar smo omejeni na računsko moč procesorske enote. Možno je namreč, da se delci izrisujejo predolgo in prekoračijo maksimalen čas izrisa, lahko pa izris upočasni nek zunanji dejavnik. Če sistem ne bo zmogel izrisati slike v določenem času, se bo število okvirjev na sekundo zmanjšalo pri tem pa se objekt časovno ne bo izrisoval pravilno. To slabost pri izrisu lahko rešimo preprosto tako, da izmerimo čas med risanjem prejšnje in trenutne slike, pridobljeno vrednost pa potem uporabimo pri računanju števila novo ustvarjenih delcev, kakor je to vidno v enačbi (3.2).

$$n' = \Delta t \, n_s \tag{3.2}$$

Tu sta:
n_s – izračunano število novih delcev na sekundo (1/s) in
Δt – časovni razmik med trenutnim in prejšnjim okvirjem (s).

V izračun števila novih delcev lahko poljubno dodamo druge spremenljivke, če želimo drugačen učinek, kot je na primer pulziranje delcev vsako sekundo ali naraščanje in padanje števila delcev ob nekem dogodku.

3.2. Lastnosti delcev

Vsak delec ima svoje edinstvene lastnosti. Med sistemi se kombinacije teh uporabljenih lastnosti lahko močno razlikujejo, vsi pa imajo skupno vsaj eno lastnost. Ta je položaj delca. Vsakega izrišemo nekje v prostoru na njegovi izračunani lokaciji.

Delci imajo različne lastnosti pri različnih objektih. Število možnih kombinacij med njimi je neskončno. Nekatere najpogosteje uporabljene so:

- trenuten položaj delca, ki je na začetku naključno določen in pogosto omejen v neko geometrijsko obliko;
- hitrost in smer, ki je ponavadi odvisna od geometrijske oblike izvora (na primer, v pravokotniku bodo imeli delci začetno smer gibanja verjetno usmerjeno pravokotno na ploskev); ponavadi se tekom časa spreminja, saj velikokrat na njih delujejo zunanje sile, najpogostejša med njimi je gravitacija;
- velikost delcev, ki je lahko ista pri vseh delcih ali pa se nastavi na začetku za vsakega posebej, lahko pa se tudi spreminja med izvajanjem učinka;
- barva ali tekstura delca ter njegova prosojnost;
- tip delca – v enem sistemu lahko imamo več različnih skupin delcev, ki izgledajo in/ali se obnašajo drugače od ostalih;
- življenjska doba, ki določa čas, po katerem delec ne bo več viden in bo verjetno popolnoma izbrisan iz sistema.

Da objekt zaživi, se morajo delcem vsaj nekatere od teh vrednosti ob izvajanju spreminjati. Lastnosti, ki so iste pri vseh, si delci lahko delijo. Potem vsakemu posamezniku ni potrebno shranjevati te informacije.

3.3. Dinamičnost

Delci sistema se ponavadi premikajo po prostoru in skozi čas spreminjajo svojo obliko, izgled in obnašanje. Nad delci lahko uporabimo privlačne in odbojne sile, rotacijo, spreminjanje prosojnosti ali druge kreativne metode, ki spremenijo izgled in položaj delca. Z določenimi omejitvami teh vrednosti ustvarimo nekakšno urejenost v sicer kaotični in naključni naravi sistema delcev. Vse je odvisno od zapisanega splošnega scenarija delcev. Pri tem lahko ti reagirajo na trke, ustvarijo nove delce, spreminjajo videz ali pa se sploh ne premikajo. Vsak delec ima sicer svojo začrtano pot, ki ima za razliko od splošne vključeno

nekaj naključnosti, kar pa mu nastavimo z njegovimi parametri.

Ko izrisujemo interaktivno grafiko, moramo pri računanju upoštevati čas med posameznimi izrisi. Nekaj razširjenih in preprostih operacij, ki premikajo delce, opišemo v sledečih odstavkih.

Sile nad delcem

Če želimo premakniti delec, mu preprosto dodamo vektor hitrosti, ki deluje na njega. Tega pred seštetjem primerno skaliramo glede na pretečen čas. Izračun je izpeljan v enačbi (3.3). Da dodamo sistemu več kompleksnosti, lahko vštejemo zraven še druge sile, ki delujejo v okolju. S tem se delec lahko premika v parabolah namesto po ravnih črtah. Potem ko premaknemo delec, glede na zunanje dejavnike posodobimo tudi silo, ki deluje nanj.

$$\mathbf{p'} = \mathbf{v}\,\Delta t + \mathbf{p} \tag{3.3}$$

Tu so:
p – položaj delca (m),
v – vektor hitrosti (m/s), ki deluje na delec in
Δt – čas (s) med izrisom trenutne in prejšnje slike.

Zadušitev hitrosti

Pri zadušitvi zmanjšamo vektor hitrosti. S tem lahko ustvarimo učinke, kot so upočasnitev delcev v tekočih snoveh in zadušitev hitrosti delca pri odboju. Kot vidimo v enačbi (3.4), zadušitev izračunamo tako, da vektor hitrosti pomnožimo s faktorjem zmanjšanja zadušitve. Če so delci umeščeni v okolje s stalno upočasnitvijo ima konstanta enoto *1/s,* zato jo še moramo zmnožiti s pretečenimi sekundami od izrisa prejšnje slike.

$$\mathbf{v'} = c\,\mathbf{v} \tag{3.4}$$

Tu je:

v – vektor hitrosti (m/s) in
c – konstanta zadušitve v območju (0, 1).

Rotacija in povečava

S to operacijo lahko transformiramo obliko delcev. To je uporabno predvsem, če delec predstavimo kot teksturo, prilepljeno na štirikotnik, ki je vedno obrnjen proti kameri. Zatem lahko ta štirikotnik povečujemo in rotiramo.

Delec lahko tudi rotiramo okoli neke druge točke. S tem lahko ustvarimo učinek rotacije delca v spiralni obliki. Vektor, ki izhaja v žarišču rotacije in kaže proti delcu, pomnožimo z rotacijsko matriko, prikazano v enačbi (3.5). Novi položaj delca pa dobimo tako, da žarišču prištejemo novi vektor.

$$\mathbf{p}' = \mathbf{R}\,\mathbf{p} = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix} \tag{3.5}$$

Tu sta:
θ – kot zasuka in
p – vektor iz točke žarišča proti položaju delca.

Barva in prosojnost

Z različno obarvanostjo lahko ustvarimo učinke, kot so utripanje delcev, pojenjanje intenzivnosti barve ali prosojnosti ob bližajočem izbrisu delca.

Trki z okoljem

Medsebojni trk med delci je računsko precej zahtevno opravilo in se ponavadi ne izvaja med takšnimi objekti, ali pa je omejeno samo na nekatere pomembnejše delce v množici. Pogosto pa v izvedbah delci trčijo ob druge objekte, ki so umeščeni v isto okolje. Ker je delcev mnogo, se uporabljajo časovno preprostejši izračuni. Najbolj znani so trki z enostavnimi objekti, kot so ravnina, kvader, krogla in črta. Druga uporabna metoda pa je trk z 2D višinskimi mrežami (*height map*), kar je uporabno pri izrisovanju tri-dimenzionalnega terena.

Trk je zaznan, ko je delec znotraj nekega objekta ali pa se je prebil skozi njegovo površino. Pri ravninah, na primer, to preverimo z ustrezno formulo površine, pri višinskih mrežah pa primerjamo višino delca z vrednostjo polja na položaju, kjer se trenutno delec nahaja.

Za izračun odboja trka poleg trenutnega vektorja hitrosti potrebujemo tudi normalo točke površine, od katere se je delec odbil. Novo pridobljeno hitrost ponavadi zadušimo, saj v realnosti trki porabijo nekaj energije, s katero je delec trčil ob drugi objekt.

3.4. Ponor delcev

Vsak posameznik množice sistema izgine ob nekem dogodku. Ob tem je delec popolnoma izbrisan, lahko pa je ponovno ustvarjen z novimi začetnimi vrednostmi. Pri nekaterih izvedbah sistema delci sploh ne izumrejo. Primeri tega so lahko izris vlaken, premikajočega tekstila ali upodobitev galaksije.

Delci so iz sistema izbrisani, ko preteče njihova življenjska doba. Možno jih je tudi izbrisati, ko je njihova hitrost zelo majhna in se opazno ne premikajo več. Tretja možnost je ta, da delci izumrejo, ko ne vplivajo več na končni videz izrisane slike (npr. so izven vidnega območja zaslona). Izginejo lahko tudi ob dotiku nekega objekta.

Ko delec umaknemo iz sistema, lahko pri tem ustvarimo nove delce ali nov podsistem, ki bo izrisoval efekt konca delca. To lahko ponazorimo s kamnom, ki ob trku razleti na več delov, ali pa s pokom mehurčka, pri čemer se lahko ustvarijo majhne kapljice vode s kratko življenjsko dobo, ki zletijo v naključno smer.

3.5. Izris

Ko so posodobljene vrednosti vseh delcev za trenutni slikovni okvir, je objekt pripravljen za končni korak življenjskega procesa. Pri izrisu se delci lahko prekrivajo, lahko so prosojni in lahko mečejo sence. Verjetno bodo v končni sceni umeščeni v neko okolje, pogosto z drugimi mnogokotniškimi modeli ali drugimi sistemi delcev.

Delce lahko na končni sliki predstavimo na različne načine. Izrišemo jih lahko z raznovrstnimi predstavitvami, med katere sodijo tudi naslednje:

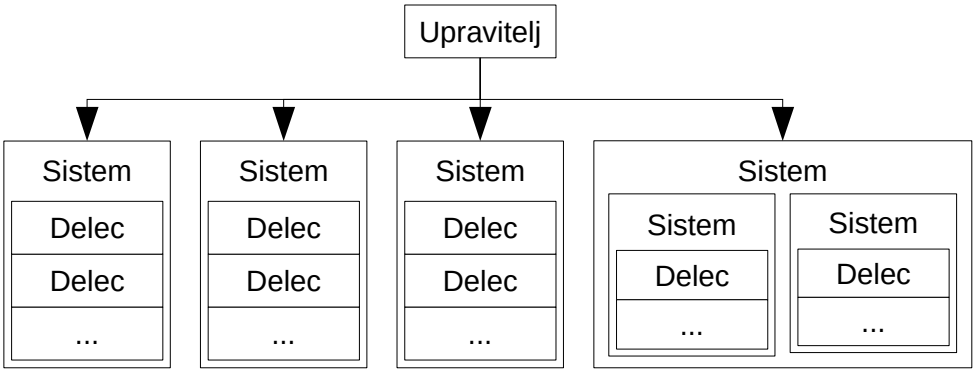
- obarvamo zaslonsko piko, ki jo delec pokriva; pogosto so ti prosojni, zato njihove barve zlijemo med sabo; to je za računalnik verjetno računsko najhitrejši način izrisa in je bil včasih bolj uporabljen;
- izrišemo črte, ki ponazarjajo smer gibanja delca;
- lahko uporabimo teksture, ki so vedno usmerjene proti kameri, te pa se lahko tudi vrtijo ali menjajo in
- kot delce lahko izrišemo tudi kompleksnejše objekte, narejene iz mnogokotnikov.

Pri interaktivnem izrisovanju slike moramo paziti na čas izrisovanja modela. Zaradi velikega števila delcev ponavadi poenostavimo izris sistema. Najpogosteje uporabljene poenostavitve so naslednje:

- delci ne mečejo sence drug na drugega,
- delci ne oddajajo svetlobe in
- ne sortiramo delcev glede na njihovo oddaljenost, pač pa skupaj zlivamo njihove barve.

3.6. Upravljanje s sistemi

Na končni sliki seveda lahko izrisujemo več sistemov hkrati. Zato je najbolje, da jih kot celoto zajamemo in združimo v en objekt, ki bo upravljal z vsemi podsistemi. Prav tako lahko nek podsistem upravlja s več svojimi podsistemi. S tem lahko sistem postane abstrakten pojem. Primer takšne hierarhije je prikazan na sliki 3.1.



Slika 3.1: Hierarhija sistemov delcev

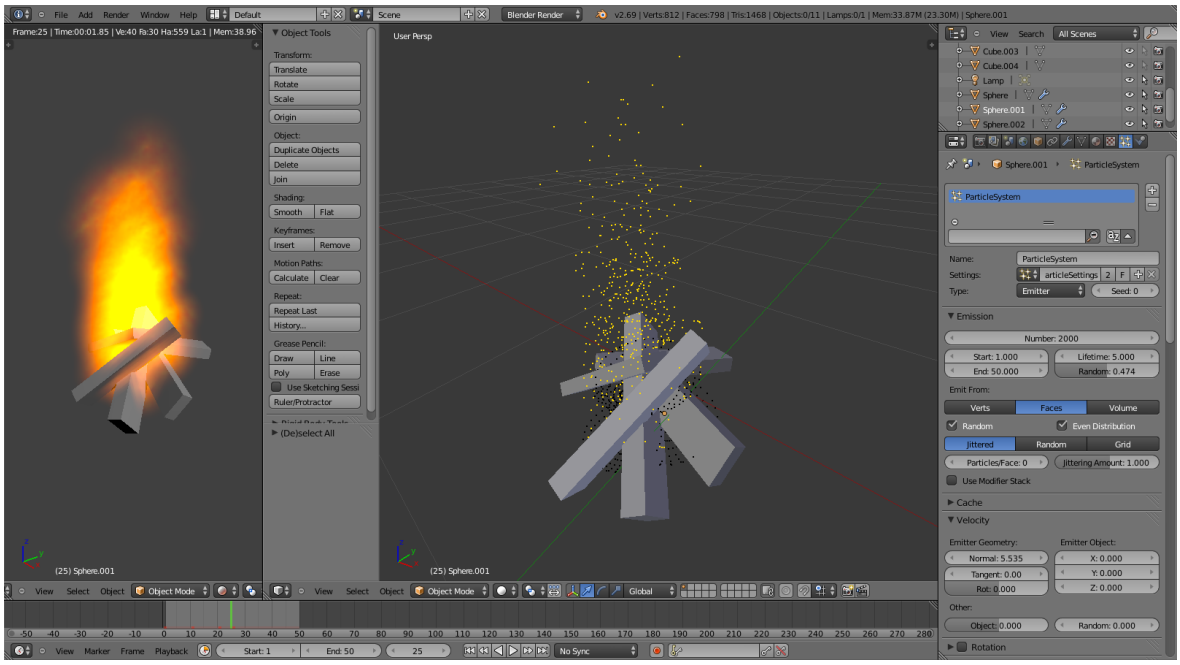
4. Praktične izvedbe

Če želimo ustvariti sistem delcev, moramo vedeti, kako bo potekal življenjski proces objekta, katere lastnosti bodo imeli delci, kakšno bo njihovo gibanje in način končnega izrisa. Obstaja že mnogo programov in orodij, ki nam poenostavijo celoten proces njihovega modeliranja.

4.1. Obstoječe rešitve

Za delo s sistemi delcev že obstajajo orodja, ki nam pohitrijo in poenostavijo njihovo modeliranje. Glede na uporabnika, ki uporablja to orodje, jih na grobo lahko razdelimo na dva dela. Razvijalci lahko uporabijo raznovrstne obstoječe programske knjižnice za stvaritev iger, simulacij ali podobnih aplikacij. Prav tako imajo orodja za delo z delci vključeni tudi nekateri programski paketi za 3D animacijo, igre, urejevalnike videov in manipulacijo slik. Te uporabljajo umetniki, ki pa ustvarjajo vizualne efekte za filme, igre ali celo pri risanju slik in retuširanju.

V mnogih programskih paketih za 3D modeliranje, načrtovanje iger ali urejanje videa lahko na preprost način ustvarjamo in oblikujemo izgled ter dinamiko sistemov delcev. Orodja se ob spremembah lastnosti izvora, delcev ali okolja odzovejo takoj ter izrišejo posodobljen in animiran objekt. Ob uporabi takšnih orodij lahko veliko hitreje in lažje modeliramo objekt, vendar pa nas ti lahko tudi omejijo v nabor funkcionalnosti, ki jih (samo) ponujajo. Nekateri znani programi, ki nam to omogočajo so Blender (na sliki 4.1), Lightwave, Cinema 4D, Maya in Photoshop.



Slika 4.1: Sistem delcev v 3D modelirniku Blender2.71

Razvijalci lahko svoj sistem delcev napišejo ter sami ustvarijo celoten življenjski cikel, lahko pa uporabijo že ponujene rešitve. Te so lahko vgrajene v nekatere pogone za igre (*game engine*) ali pa v aplikacije za delo z vizualnimi učinki. Primeri teh aplikacij so Unity3D, Game Maker in Havok. Seveda pa lahko sisteme delcev razvijalci (specifično programerji) izdelajo v celoti svoje, še posebej v primeru če nam dostopne rešitve ne ponudijo zmogljivosti, ki jih potrebujemo ali pa želimo nestandardne učinke ali vedenje objekta. Tudi v tem primeru lahko vsaj za določen del sistema delcev uporabimo že obstoječ produkt, največkrat kakšne knjižnice za simulacijo fizikalnih pravil, ki jih potrebujemo pri premikanju ali odboju.

4.2. Primeri življenjskih ciklov

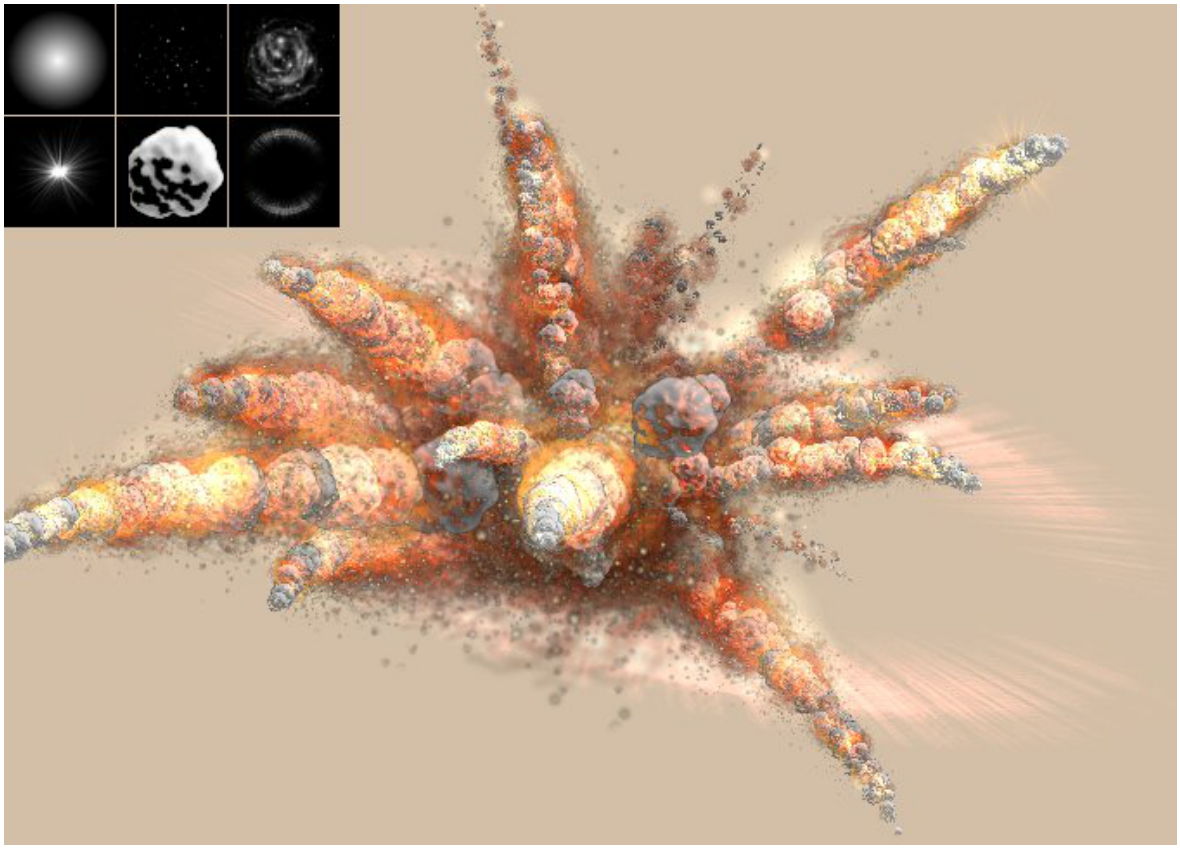
S to tehniko je možno ustvariti pojave, kot so eksplozije, padajoči listi, tornado, zvezde na nebu in iskre. Vsak primer ima svoje lastnosti, način interakcije z okoljem in življenjski cikel, vendar pa si večinoma med seboj delijo koncepte, kot so gravitacija, odboj in rotacija. Sledi nekaj splošno opisanih primerov, kako lahko uporabimo metodo za izris različnih pojavov.

Ena izmed najbolj osnovnih in enostavnih izvedb sistema oddaja delce v neki izvorni točki ali določenem območju, iz katerega ti potem letijo v različne smeri. Delci sistema se neprestano ustvarjajo z neko življenjsko dobo in nadaljujejo pot v podani smeri, medtem ko jih sila gravitacije pospešeno vleče proti tlom. Obarvani delci lahko skozi čas menjajo svojo barvo in velikost, po pretečeni življenjski dobi pa izginejo. S takšnim postopkom lahko oponašamo dim, delovanje vodometa, izbruha vulkana ali vodne slapove. Sistemu lahko dodamo tudi zmožnost zaznavanja trkov ter posledično odboja posamičnih delcev od okolja, v katerega je objekt umeščen.

Pri nekaterih pojavih ni nujno, da opazimo njihov izvor, saj se lahko delci ustvarjajo po vsem okolju ali pa je izvor zelo velik. Na padavine, kot so dež in sneg vpliva gravitacija ter smer in moč vetra. Ustvarijo se v oblaku, za kar lahko uporabimo ravnino, na kateri se pojavijo delci in drvijo proti tlom. Ob udarcu na površje izginejo, dežne kaplje pa še lahko ustvarijo pljusk. Po celotnem okolju se lahko premikajo tudi podvodni mehurčki ali pa listje v gozdu.

V prvem primeru iz prve publikacije o sistemih delcev je opisana gradnja scene, kjer opustošen planet z razširjajočim ognjenim zidom spremenijo v naseljivega. Ta primer je bil uporabljen v že omenjenem filmu. Iz točke udarca, ki smo ga že videli na sliki 2.1, se je ogenj začel krožno širiti po planetu. Sistem delcev je imel dvo-nivojsko hierarhijo. Iz točke udarca v planet so se ustvarjali delci, ki so bili tudi posamezni sistemi delcev in so ustvarjali ognjene delce.

Delci pri eksploziji se pojavijo v eni izvorni točki, ki je v samem centru eksplozije. Nato hitro zletijo v naključno določeno smer, kmalu zatem pa se uničijo. Gravitacija jih vleče k tlom. Za sabo puščajo druge delce, ki so izrisani kot ogenj in dim. S časom spremenijo barvo iz svetlo-rumene v rdeče-oranžno. Delci lahko imajo tudi različne teksture za pristnejši izgled. Eksplozija se ne ponavlja, zato se delci tudi ne ustvarjajo ponovno. Izgled lahko vidimo na sliki 4.2.

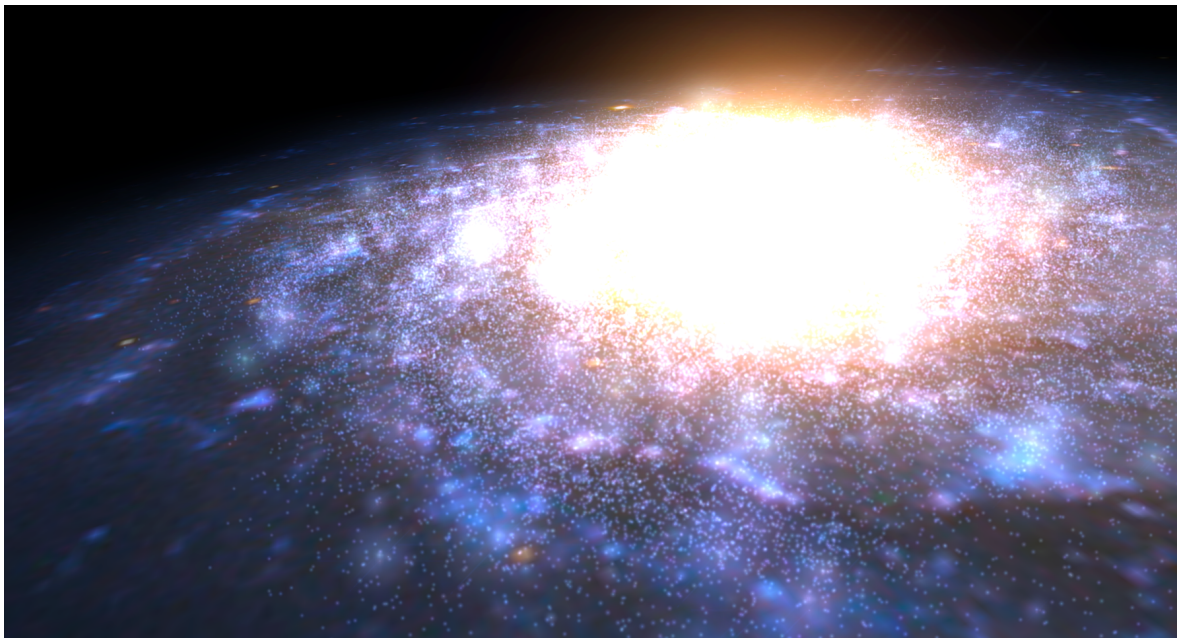


Slika 4.2: Eksplozija delcev ter teksture uporabljene pri izrisu [Wikipedia, 2014]

Trava se od ostalih razlikuje po tem, da se položaji posameznih delcev ne spreminjajo. Ti se tudi ne ustvarjajo ponovno, ampak so definirani samo enkrat, pred prvim izrisom. Ob tem se jim nastavi geometrija in tekstura. Posamezne bilke se še lahko upogibajo ob tem, ko piha veter. Večjo vlogo ima pri tem izris delcev. Če želimo izgled bolj podoben risanim filmom, lahko iz tal izrisujemo dolge trikotnike, obarvane z različno zeleno barvo. Če pa želimo bolj realističen videz, kot na sliki 4.3, lahko na kvadrat prilepimo teksturo s šopom trave, ter jih po pokrajini »posejemo« v različnih smereh.

Zvezde v galaksiji se gibajo okoli centra v krožni obliki. Delci so ustvarjeni samo enkrat. V središču galaksije pa so nanizani bolj gosto. Nekatere galaksije, kot je naša, imajo obliko spirale, zato ji tudi lahko delce pogosteje razporedimo po nekaj naključno ustvarjenih spiralnih krakih. Zvezde lahko še utripajo. Položaje zvezd lahko črpamo tudi iz obstoječih podatkov in tako vizualiziramo našo galaksijo, kot vidimo na sliki 4.4.

Slika 4.3: Uporaba delcev pri izrisu travnih bilk [Reeves, 1983]



Slika 4.4: Spletna interaktivna galaksija stars.chromeexperiments.com [1. 9. 2014]

5. Animacija ognja

Naš sistem delcev je napisan v programskem jeziku C++. Celotna aplikacija se izrisuje s pomočjo programskega vmesnika OpenGL. Pri prikazu okna in upravljanju z uporabniškimi vhodnimi dogodki skrbi večpredstavnostna programska knjižnica SDL2[2]. Za izris grafičnega vmesnika se uporablja 2D vektorska grafična knjižnica NanoVG[3]. Uporabljeno je bilo razvojno okolje Visual Studio C++ 2010 Express.

Celotna izvedba sistema delcev je izpeljana v dvodimenzionalnem prostoru.

5.1. Programski vmesnik OpenGL

Dandanes imajo vsi moderni računalniki (že tudi mobilni telefoni) vgrajene namenske grafične procesne enote, ki pospešijo izris grafike uporabniškega vmesnika, računalniških iger, video posnetkov in drugih multimedijskih vsebin. Do teh zmogljivosti lahko dostopamo z uporabo različnih grafičnih programskih vmesnikov, najbolj razširjena sta OpenGL in pa Direct3D. Medtem ko je slednji namenjen samo platformi Windows, je OpenGL prost standard in je podprt tudi na drugih platformah.

Že kar nekaj let grafične kartice podpirajo tako imenovan programljiv cevovod (*programmable pipeline*). Ta nam glede na starega funkcijskega nudi več zmogljivosti in hitrejše izrisovanje grafike. V primeru OpenGL API-ja najprej v namenskem jeziku GLSL, ki je precej podoben C-ju, napišemo programe za grafično kartico (*shaders*). Zatem izvorno kodo pošljemo grafičnemu čipu, ki jih prevede in izvede risanje, ko mu to ukažemo.

2 Simple DirectMedia Layer (https://www.libsdl.org/)

3 NanoVG (https://github.com/memononen/nanovg)

5.2. Kamera

Celotna aplikacija uporablja različne koordinatne sisteme. Za pretvorbo med njimi skrbi razred *Camera*. Njena glava naloga je premik in povečava celotne 2D scene, v kateri se izrisujejo delci. Matriko, ki jo pridobimo iz te transformacije, uporabimo v GPU-ju pri izračunu položaja delcev.

Transformacija mora vidne delce na sceni premakniti in skalirati v območje [-1, 1] na obeh oseh. Takšen je namreč koordinatni sistem, v katerem OpenGL izrisuje sceno. Kamera izračuna transformacijo iz treh parametrov. Ti so:

- velikost okna (širina in višina) v zaslonskih pikah,
- vektor premika celotne scene in
- skaliranje (manjšnje te vrednosti pomeni bližanje delcem).

Za izračun končne matrike najprej potrebujemo velikost ene enote scene v koordinatnem sistemu okna. Kot vidimo v enačbi (5.1), to izračunamo tako, da povprečje med širino in višino okna delimo s skalirnim faktorjem.

$$a = \frac{w + h}{2z} \tag{5.1}$$

Tu je:
w in h – velikost okna (širina in višina, ki sta obe večji od 0) in
z – povečava scene (vrednost je večja od 0).

Zatem potrebujemo skalirni faktor za obe osi, ki ju izpeljemo v enačbah (5.2) s pomočjo preslikave razmerij med koordinatnimi sistemi. Pri tem je v enačbi ciljna velikost vedno 2 enoti (OpenGL območje).

$$\frac{a}{w} = \frac{s_x}{2}, \quad s_x = \frac{2a}{w}; \qquad \frac{a}{h} = \frac{s_y}{2}, \quad s_y = \frac{2a}{h}; \tag{5.2}$$

Za pridobitev končnega rezultata najprej premaknemo točke, pri čemer uporabimo vektor premika scene. Potem točke skaliramo z vrednostmi, ki smo jih pridobili iz enačbe (5.2).

Matriki teh dveh transformacij zmnožimo. S tem smo dobili končno matriko (5.3), ki jo pozneje uporabimo pri računanju položajev delcev v programih za GPU.

$$\mathbf{M} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & s_x t_x \\ 0 & s_y & s_y t_y \\ 0 & 0 & 1 \end{pmatrix} \tag{5.3}$$

Tu je:
t – vektor premika scene.

Pri implementaciji, vidni iz izpisa 5.1, matriko shranimo v pomnilnik zaporedno po stolpcih (*column-major order*), ker tako zahteva OpenGL.

Izpis 5.1: Izračun transformacijske matrike

```
mat3 Camera::GetTransformMatrix() const {
    // skaliranje
    float a2 = (window.w + window.h) / zoom;
    float sx = a2 / window.w;
    float sy = a2 / window.h;

    // premik
    float stx = offset.x * sx;
    float sty = offset.y * sy;

    return mat3(
          sx,   0, 0,
           0,  sy, 0,
         stx, sty, 1
    );
}
```

Poleg izračuna transformacijske matrike lahko iz razreda pridobimo pretvorjeno točko iz koordinatnega sistema scene v koordinatni sistem okna in nazaj. To je uporabno pri interakciji s kazalčnimi vnosnimi napravami, kot je računalniška miška. Uporabno pa je tudi pri izrisovanju uporabniškega vmesnika, ki pa se izrisuje na osnovi koordinat zaslonskih pik z žariščem levo zgoraj. Skalirni faktor je nujen tudi, ko računamo velikost izrisujočih delcev.

5.3. Nadzor nad sistemi

Vsi izvori delcev so shranjeni v objektu razreda *Manager*. To je glavni objekt na naši sceni. Zadolžen je za stvarjenje in upravljanje z vsemi izvori. Določa tudi zaporedje izrisa sistemov. Pred vsako upodobitvijo sistema najprej izvorom ukaže pravilno časovno posodobitev vseh delcev, zatem pa jih po vrsti izriše.

Temu objektu lahko nastavimo globalni lastnosti, ki jih uveljavi nad vsemi delci v sistemih. Ti dve sta:

- vektor hitrosti vetra in
- daljice, od katerih se delci lahko odbijejo.

5.4. Shramba delcev

Delcev je v sistemu ponavadi veliko, zato jih tudi moramo učinkovito shraniti v pomnilnik ter jih pozneje priklicati in upravljati z njihovimi vrednostmi. Izbrati si moramo podatkovno strukturo, ki bo v našem primeru delovala najbolj učinkovito. Za delce v sistemu največkrat potrebujemo dinamično strukturo, ki ji lahko spreminjamo velikost in hitro dodajamo in brišemo delce iz njih. Pomembna pa je tudi zaporedna hitrost dostopanja do podatkov. V glavnem poznamo dva najpogostejša načina shrambe delcev.

Najpreprostejša je podatkovna struktura vektor, ki pa je polje enakih podatkovnih tipov. Ti so shranjeni v pomnilniku zaporedno, eden za drugim. Pri vektorju lahko dostopamo do katerega koli podatka v takojšnjem času, slaba lastnost pa je počasnejša sprememba njegove velikosti in pa vstavljanje na sredino strukture.

V povezan seznam se shranjujejo delci nekoliko drugače. Vsak delec ima poleg svojih podatkov še kazalec na naslednjega, lahko tudi na prejšnjega. Zadnji delec pač kaže na prazen pomnilniški prostor. S tem lahko hitro in dinamično dodajamo in odstranjujemo delce. Čez delce se sprehodimo tako, da izrišemo prvi delec, nato pridobimo naslednjega in to ponavljamo vse do zadnjega. Ta podatkovna struktura se obnese odlično, če ne potrebujemo delcev zaporedno v pomnilniku.

Za čim hitrejši izris na grafični kartici moramo podati programskemu vmesniku kazalec na podatke in njihovo število. Pri povezanem seznamu so podatki v pomnilniku shranjeni raztreseno, zato bi morali za vsak delec posebej klicati funkcijo za izris. To je v nasprotju z dobro znano prakso, da za hitrejše izvajanje skušamo čim bolj zmanjšati število klicev funkcij OpenGL za izris objektov.

Ker se morajo v tej izvedbi posodobljeni podatki pred izrisom prekopirati iz glavnega pomnilnika v pomnilnik grafičnega procesorja, je uporabljena struktura vektor z nekaj spremembami. Podatkovna struktura je uporabljena zaradi njenega načina shrambe podatkov. S tem lahko najhitreje prekopiramo (prenesemo) podatke v GPU.

Novi delci se v strukturo dodajajo na konec. Ko pa je treba nek delec izbrisati iz sistema, je verjetno ta nekje na sredini podatkov. Če tega izbrišemo, bo nastala neuporabljena luknja, ki jo moramo najhitreje zapomniti. Namesto da bi pomaknili vse sledeče podatke za eno mesto nazaj, raje na to mesto prekopiramo delec iz konca strukture. Nato samo še zmanjšamo velikost strukture za en delec.

5.5. Izvor delcev

V naši izvedbi sistema razred *Emitter* predstavlja izvor delcev. Njegova naloga je shramba vseh teh delcev v podatkovno strukturo, tvorba novih, posodobitev njihovih vrednosti ter končni izris. Nastavijo se mu lahko različne lastnosti, ki vplivajo na obnašanje in videz sistema. Razdelimo jih lahko na več skupin.

Najprej je delec ustvarjen v sistem. Na njegove začetne lastnosti vpliva kar nekaj parametrov:

- objekt, ki ustvari nov začetni položaj delca;
- objekt, ki tvori nov vektor začetne hitrosti in smeri;
- spremenljivki za najkrajši čas, kolikor ga delec mora preživeti v sistemu ter razpon časa med najkrajšim in najdaljšim možnim časom trajanja delca;
- približno število novih delcev, kolikor se jih lahko ustvari na sekundo;
- razpon frekvence nihanja delcev in
- razpon amplitude začetnega nihanja delcev.

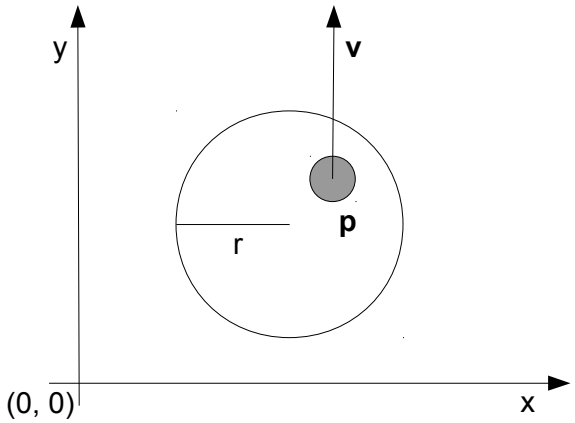
Delcem se pred vsakim izrisom posodobijo nekatere vrednosti. Lastnosti, ki vplivajo na gibanje posameznikov, so:

- teža teh delcev, ki se odšteje od sile gravitacije, tako da lahko delci odletijo tudi navzgor;
- faktor upočasnitve delcev (upočasni se vektor pospešitve) in
- faktor zmanjšanja amplitude nihanja posameznega delca (na sekundo).

Na končen izris vpliva več lastnosti. Te so:

- velikost vseh delcev (ta vrednost se pred končnim izrisom preračuna, tako da se še upošteva povečava scene);
- barva in prosojnost delcev;
- način zlivanja barv med delci (prebarvaj ali seštej barve) in
- način izrisa delcev (krog ali žareč prehod).

Za tvorbo začetnih položajev in vektorjev hitrosti smo ustvarili različne objekte, ki nam vračajo naključne ali konstantne vrednosti. Vsi njihovi razredi si delijo skupen vmesnik, tako da jih lahko med seboj menjujemo med tem ko teče aplikacija. Eno od možnih kombinacij objekta, ki ustvari položaj in silo, lahko vidimo na sliki 5.1.



Slika 5.1: Primer začetnega položaja delca v krogu z nespremenljivim vektorjem hitrosti

Nove delce lahko ustvarimo v različnih območjih. Ti so lahko:

- stalna točka,
- krog s podanim polmerom ali
- daljica z žariščem in vektorjem do druge točke.

Inicializacija vektorja hitrosti je ustvarjena na podoben način. Lahko je:

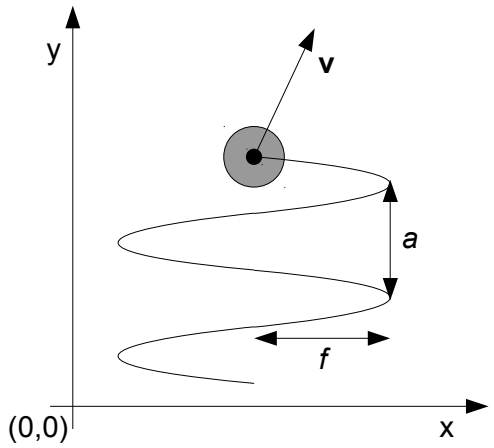
- konstanten vektor,
- naključni vektor v krogu ali
- vektor v določeni smeri z naključno dolžino.

5.6. Lastnosti delcev

Posamezni delci nimajo tako veliko lastnosti kot izvor. Potrebujemo jih zato, da ustvarimo nekaj naključnosti, da se vsi ne gibajo po isti poti. Te lastnosti so:

- trenutni položaj delca v 2D prostoru;
- trenutna sila, ki nakazuje smer, v katero se delec giba in hitrost v tej smeri;
- časovni zaznamek za izbris;

- trenutna amplituda;
- trenutni kot nihanja z njegovo povečavo na sekundo in
- izračunan odmik delca zaradi nihanja (iz prejšnje posodobitve).



Slika 5.2: Lastnosti delca – položaj in smer ter velikost amplitude in frekvence nihanja

5.7. Gibanje delcev

Delci ognja se v naši izvedbi gibajo s svojo začetno hitrostjo, istočasno pa nihajo svoj položaj v eno ali drugo stran. Na posamezen delec delujejo tudi zunanje sile, ki ga pospešijo. Ta pospešitev spremeni hitrost delca, hitrost pa spreminja njegov položaj.

Delci imajo vse od svojega nastanka določeno svojo hitrost in smer. Vse do njihovega konca se stalno gibajo. Da pravilno premaknemo delce ob vsaki posodobitvi uporabimo že izpeljano enačbo (3.3). Pred tem vektorju hitrosti prištejemo zunanje sile, ki delujejo nanj. Ta vektor tudi zatremo, če imamo nastavljeno upočasnitev delcev. Pri tem si pomagamo z enačbo (3.4).

Vsak delec niha z določeno amplitudo in periodo. Za njen izračun potrebujemo več spremenljivk, ki smo jih že našteli v prejšnjem poglavju. Najprej zvišamo trenutni kot nihanja za del periode, ki je odvisna od časa, ki je pretekel od prejšnjega izrisa. Iz tega kota nato izračunamo sinus vrednosti, ki ga zmnožimo z amplitudo delca. Izračun odmika lahko

vidimo v enačbi (5.4). Dobimo rezultat, ki ga odštejemo od rezultata iz prejšnje posodobitve. Razliko na koncu prištejemo k x osi položaja delca.

$$o = a \sin(\theta \lambda) \tag{5.4}$$

Tu so:

o – izračunan odmik,
θ – trenuten kot,
λ – velikost periode in
a – amplituda.

Celotna posodobitev sistema najprej vsak delec pregleda, če ga je že potrebno izbrisati. Če je delec še živ, se izračuna njegov nov položaj. Na koncu se še ustvari nekaj novih delcev. Koda za posodobitev delcev z vsemi opisanimi koncepti je napisana v izpisu 5.2.

Izpis 5.2 (1. del): Posodobitev delcev

```
void Emitter::Advance(Uint32 timeMark, float sec, vec2 force,
const std::vector<LineF> &lines)
{
    vec2 addForce = force * sec;
    float slow = 1.0f - (slowing * sec);

    Particle *particle = particles.data();
    Particle *limit = particle + particles.size();
    while (particle < limit)
    {
          Particle p = *particle;
```

Izpis 5.2 (2. del)

```
		// izbris
		if (timeMark > p.end) {
			*particle = *(--limit);
			particles.pop_back();
			continue;
		}

		// uveljavi sile
		p.force -= addForce;
		p.force *= slow;
		p.position += p.force * sec;

		// nihanje
		p.oAngle += p.oIncrement * sec;
		float osc = sinef(p.oAngle) * p.oAmplitude;
		p.position.x += osc - p.oOffset;
		p.force.x += osc - p.oOffset;
		p.oOffset = osc;
		p.oAmplitude *= 1.0f - (oAmpSlow * sec);

		*particle = p;
		particle++;
	}

	// dodaj nove
	int msec = (int)(sec * 1000.0f + 0.5f);
	unsigned nNew = (unsigned)(sec * pGenSec + 0.5);
	for (unsigned i = 0; i < nNew; i++)
```

Izpis 5.2 (3. del)

```
    {
        Particle p;
        Init(p, timeMark);

        // napreduj
        if (0 < msec) {
            float seconds = (rand() % msec) / 1000.0f;
            p.force -= force * seconds;
            p.position += p.force * seconds;
        }
        particles.push_back(p);
    }
}
```

5.8. Zaznavanje trka in odboj

Ob vsaki osvežitvi podatkov moramo preveriti morebitne trke delcev z daljicami, od katerih se odbijajo. Zaradi preprostosti si pot delca med prejšnjim in trenutnim časom omislimo kot daljico med prejšnjim in trenutnim položajem delca. Torej moramo med vsako potjo delca in daljico okolja preveriti, če se ti dve sekata. To lahko testiramo s pomočjo vektorskega produkta. Z njim lahko ugotovimo, če so oglišča ene daljice na nasprotnih straneh premice, na kateri leži druga daljica.

Ob zaznanem trku moramo delec odbiti od podlage in ga usmeriti v drugo smer. Natančneje, vektor poti delca moramo preslikati čez vektor normale daljice. V tem primeru normala predstavlja enotski vektor, ki je pravokoten na površino, od katere je potrebno delec odbiti.

Kakor vidimo na sliki 5.3, normalo površine dobimo z zasukom njenega vektorja za 90° v katerokoli smer, ki ga nato delimo z njegovo dolžino. Smer normale ni pomembna zato, ker ne vpliva na končni izračun odboja. Formula za zasuk vektorja se pri kotu 90° precej poenostavi, kot vidimo v enačbi (5.5).

$$\begin{aligned} n_x \cdot |\mathbf{p}| &= p_x \cdot \cos(90°) - p_y \cdot \sin(90°) = -p_y \\ n_y \cdot |\mathbf{p}| &= p_x \cdot \sin(90°) + p_y \cdot \cos(90°) = p_x \end{aligned} \tag{5.5}$$

Tu sta:

p – dvodimenzionalni vektor podlage in

n – izračunana normala vektorja podlage.

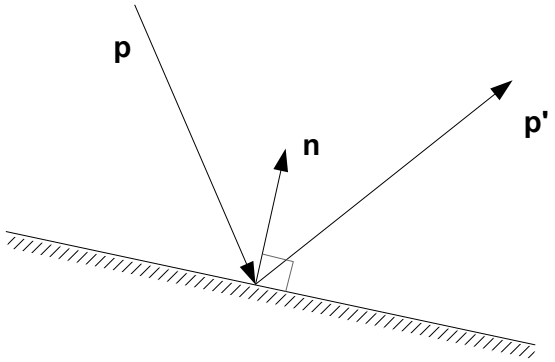
Zdaj lahko izračunamo preslikavo. To storimo tako, da najprej s skalarnim produktom projeciramo originalni vektor na normalo. Dobimo dolžino projekcije, ki jo podvojimo, nato pa jo zmnožimo z normalo. Kar s tem pridobimo, je vektor v smeri normale z dvakratno dolžino te projekcije. Od tega še samo odštejemo podani vektor in dobimo pravilno preslikavo, kot v enačbi (5.6).

$$\mathbf{p}' = 2(\mathbf{n} \cdot \mathbf{p})\mathbf{n} - \mathbf{p} \tag{5.6}$$

Tu sta:

p – vektor, ki ga odbijemo od podlage in

n – normala podlage.



Slika 5.3: Preslikava vektorja skozi vektor normale

Ker praktično nobena stvar v naravi po odboju nima enako velike sile kot pred njim, ta vektor zadušimo z neko vrednostjo. Kot smo že videli v enačbi (3.4), vektor zmnožimo z vrednostjo v območju med 0 in 1. S tem upočasnimo silo, ki deluje na delec. Če bi uporabili vrednost večjo kot 1, bi delec bil nenaravno hiter, negativne vrednosti zadušitve pa bi delec zabile nazaj v ploskev in bi se zaradi takojšnjega ponovnega odboja to dogajalo v neskončnost, vsaj dokler ni ta izbrisan.

5.9. Izris delcev z OpenGL

Za izris delcev je uporabljen najpreprostejši možen element pri risanju z OpenGL. To je točka, ki ji še dodatno nastavimo velikost.

Vsi položaji točk se pošljejo grafični procesni enoti, ki jih ta s pomočjo aktivnega progama, ki teče na njem, obarva z določenimi parametri. Programi so sestavljeni iz dveh delov:

- prvi del pravilno transformira položaj delca (*vertex shader*),
- drugi obarva posamezne piksle (*fragment shader*).

V prvi stopnji izrisa se nad vsako točko izvede »vertex shader«. Njegova naloga je izračunati končni položaj posameznih točk na sceni. Ta vrednost se mora shraniti v 4D vektor *gl_Position.*

V našem primeru, ki ga lahko vidimo v izpisu 5.3, kot globalno spremenljivko uporabimo matriko *u_matrix*, ki smo jo izračunali v enačbi (5.3). Položaj delca pridobimo iz vhodne spremenljivke *a_position*. Najprej ga spremenimo v homogene koordinate, da ga lahko zmnožimo z vhodno matriko. Pridobljeno vrednost shranimo v izhodno pozicijo. Ker lahko z OpenGL-om izrisujemo 3D grafiko, ima ta spremenljivka 4 elemente, naša transformirana točka pa enega manj. Zadnji element izračunane pozicije ne predstavlja globine (osi *z*), ampak vrednost, ki mora biti enaka številu ena[4]. Zato zadnji element 3D

4 Če shranjena vrednost ni enaka 1, se vsi elementi vektorja delijo s to vrednostjo, tako da postane 1.

točke prestavimo v zadnji del vektorja *gl_Position*, delce pa izrišemo kar na globini 0. Pri tem so točke, ki bodo vidne na zaslonu, vedno v območju [-1, 1] na vseh treh oseh.

Izpis 5.3: GPU program, ki izračunava položaje delcev

```
#version 150
uniform mat3 u_matrix;

attribute vec2 a_position;

void main()
{
    vec3 p = u_matrix * vec3(a_position, 1.0);
    gl_Position = vec4(p.x, p.y, 0.0, p.z);
}
```

Na drugi stopnji programa »fragment shader« obarva zaslonske pike na naši sceni. Ta stopnja se ponavadi izvede večkrat, kot prejšnja, zato je pri tej bolj pomembna hitrost izvajanja. V našem primeru se za vsak delec pozicija izračuna samo enkrat. Ker pa so ti mnogokrat izrisani čez več zaslonskih pik, se mora druga stopnja izvršiti večkrat za en delec.

Izvor delcev ima nastavitev, ki določa ali se bodo posamezniki izrisali kot obarvan krog ali pa s postopnim žarečim prehodom (*radial gradient*) barve v prosojno. Za to tudi potrebujemo dva GPU programa. Oba sicer uporabljata isti izračun pozicije vendar se delci obarvajo drugače.

V prvem primeru, ko izrišemo prosojen krog, je program zelo preprost, saj samo nastavi barvo kot izhodno spremenljivko, ki jo imamo nastavljeno za delce enega sistema.

V drugem primeru pa je izris delca malce bolj kompleksen. Program izriše delec z žarečim prehodom iz popolnoma prosojne do dejanske barve delca. Program naprej s pomočjo

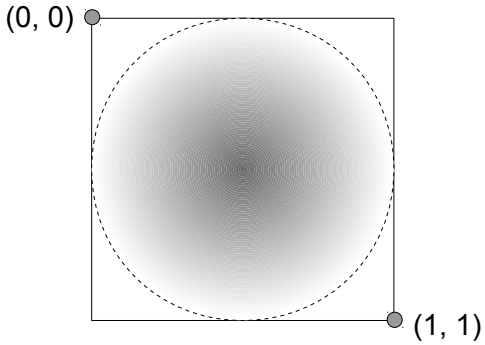
vgrajene spremenljivke *gl_PointCoord*[5] izračuna oddaljenost trenutnega piksla od središča. Nato iz tega izračuna njegovo prosojnost, ter jo na koncu zmnoži s prosojnostjo iz barve delca. Kot je vidno iz slike 5.4, je rezultat prehod med barvo delca *u_color* na sredini in med prosojno barvo na robovih kroga. Dejanski GPU program je priložen kot izpis 5.4.

Izpis 5.4: GPU program, ki obarva delce s prehodom

```
#version 150

uniform vec4 u_color;

void main()
{
    float dist = length(gl_PointCoord - vec2(0.5));
    float alpha = 1.0 - (dist + dist);
    gl_FragColor = vec4(u_color.rgb, u_color.a * alpha);
}
```

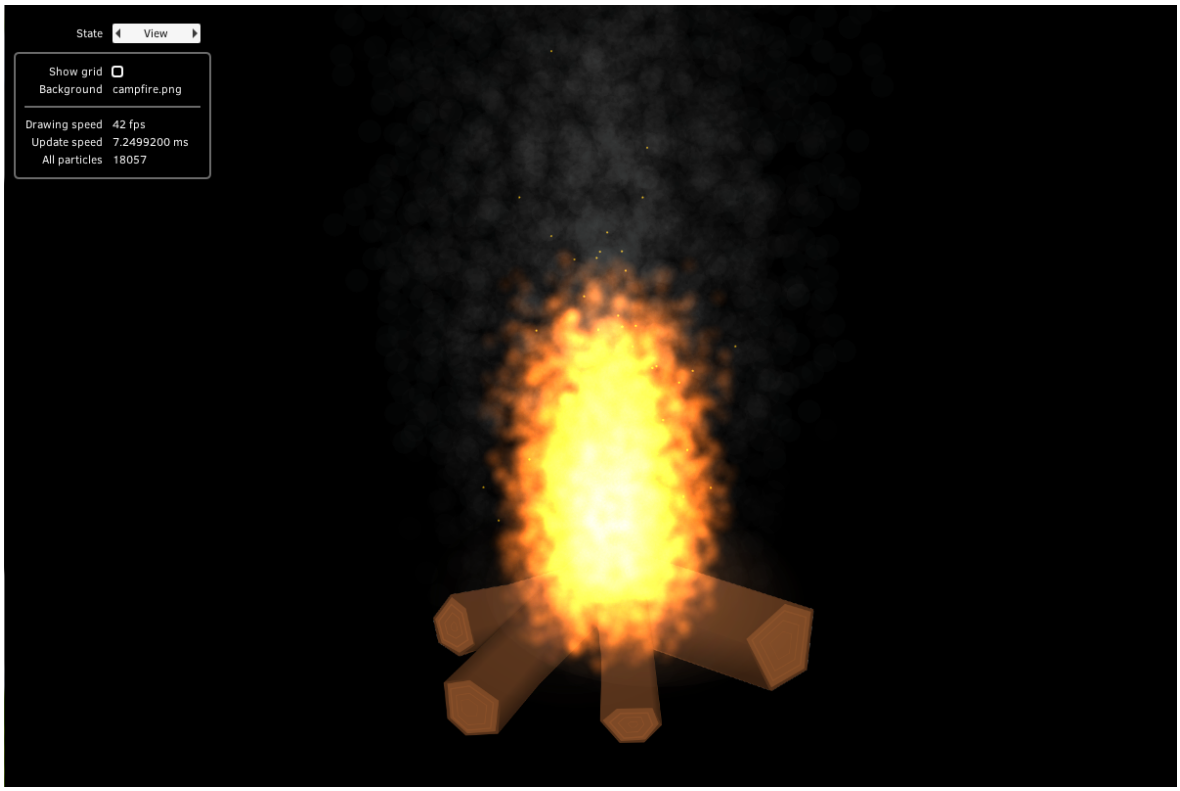


Slika 5.4: Izris delca s prehodom

5 Da je spremenljivki *gl_PointCoord* dodeljena vrednost, moramo pred izrisom vklopiti zastavico *GL_POINT_SPRITE* s funkcijo *glEnable()*.

5.10. Uporabniški vmesnik

Uporabniški vmesnik aplikacije je razdeljen na dva glavna dela. Čez celotno okno se v ozadju upodablja scena. Tu se izrisujejo posamezni delci iz vseh sistemov, prav tako pa koordinatna mreža in ozadje scene, če smo ju nastavili. S klikom in potegom srednjega gumba na miški se lahko premikamo po sceni, z vrtenjem koleščka pa povečujemo in oddaljujemo pogled. Na levi strani se izrisuje uporabniški vmesnik, kjer so prikazane različne informacije in kjer lahko spreminjamo lastnosti različnih objektov v programu. Končen izgled lahko vidimo na sliki 5.5.

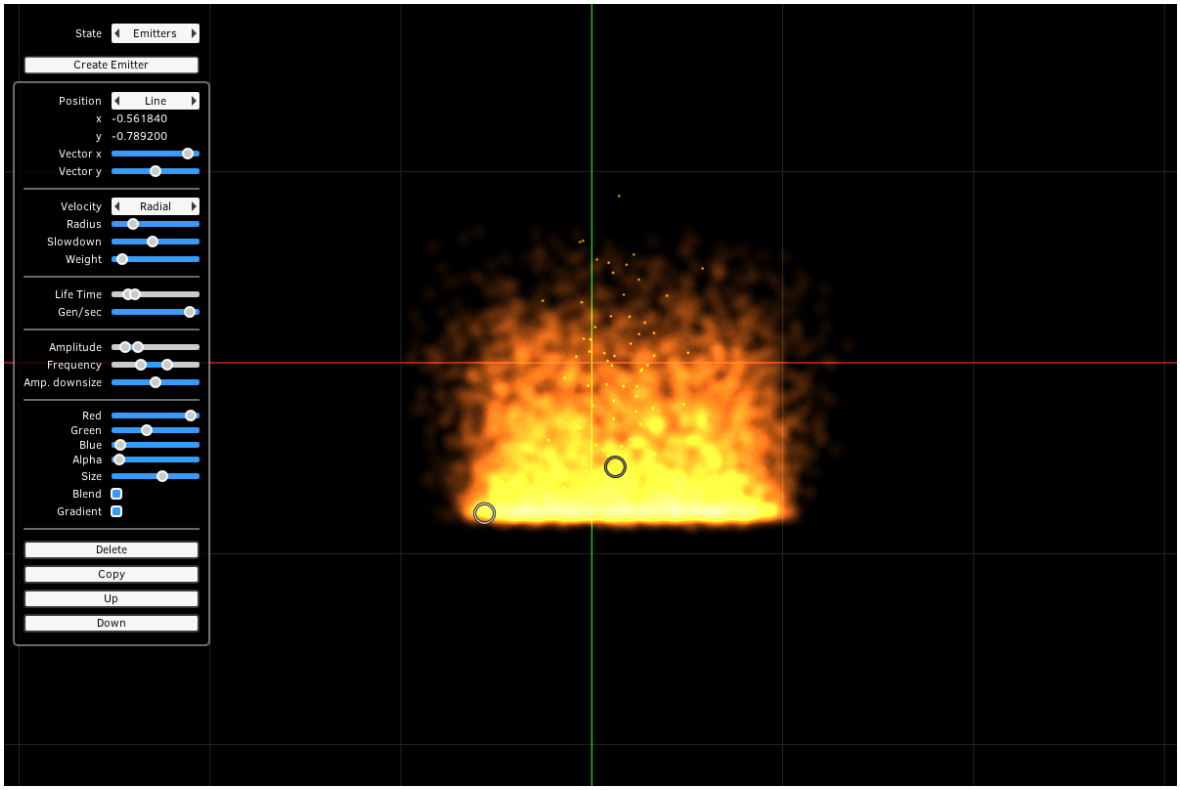


Slika 5.5: Izris ognja, isker in dima.

V programu lahko urejamo različne parametre in objekte. S prvim gradnikom levo zgoraj v uporabniškem vmesniku se lahko pomikamo v različna stanja urejanja naše scene. Ta so:

- pregled scene, kjer lahko vidimo število vseh izrisanih delcev, število izrisanih slik na sekundo (FPS) in hitrost izračuna novih delcev pred izrisom, prav tako pa lahko preklapljamo vidnost koordinatne mreže in vidimo ime odprte slike ozadja;

- urejevalnik izvorov, kjer lahko urejamo parametre za upravljanje delcev, izbrišemo izvore, jih kopiramo in premikamo po sceni;
- urejevalnik nastavitev moči vetra, ki odnaša delce in
- urejevalnik odbojnih daljic, ob katere lahko delci trčijo.



Slika 5.6: Primer urejanja dveh izvorov delcev, ki ustvarjata ogenj

Najbolj uporaben in kompleksen del uporabniškega vmesnika je urejevalnik izvorov, ki ga lahko vidimo na sliki 5.6. Tu lahko premikamo izvore s klikom in potegom miške po sceni. Z uporabniškim vmesnikom lahko ustvarimo nov izvor, obstoječe pa lahko izbrišemo, jih kopiramo ali ustvarimo nove. Prav tako lahko izvore pomikamo nad ali pod ostale, tako da lahko izrisujemo končni vizualni učinek v pravilnem vrstnem redu. Osrednji del je namenjen spremembi različnih parametrov, ki vplivajo na izris in obnašanje sistema.

Urejevalnik vektorja vetra je zelo preprost. S klikom gumba in potegom miške spreminjamo vektor hitrosti vetra.

Z urejevalnikom trkov lahko spreminjamo daljice, od katerih se delci odbijajo. Z levim

gumbom in potegom miške lahko spreminjamo položaje obstoječih oglišč daljic. Z desnim klikom ustvarimo novo oglišče, če ponovimo to na drugem mestu, ustvarimo novo daljico. Če pa z desnim miškinim gumbom kliknemo na obstoječa oglišča črt, potem to daljico izbrišemo.

5.11. Pohitritev izvajanja

Pri osveževanju položaja delcev smo naredili ključno pohitritev pri računanju za čim hitrejše izvajanje scene. To smo uveljavili pri izračunu nihanja delca. Tu potrebujemo funkcijo, ki nam izračuna sinus podanega kota. Najprej smo uporabili funkcijo *sin()* iz standardne knjižnice jezika C++. Ob tem se je hitrost izvajanja zmanjšala za kar tretjino. Ker želimo čim hitrejše izvajanje našega sistema in hkrati ne potrebujemo velike natančnosti standardne funkcije, lahko naredimo preprosto pohitritev. Vrednosti sinus funkcije na eni valovni dolžini[6] smo shranili v vpogledno tabelo z neko velikostjo. Ko želimo pridobiti približno vrednost, izračunamo indeks v tabeli, kjer je shranjena vrednost. Slabost pri tej funkciji je seveda ne preveč dobra natančnost. Kot pa je iz tabele 5.1 razvidno, se nam je hitrost izračuna delcev precej zmanjšala, prav tako pa ne opazimo razlike ob animaciji.

Tabela 5.1: Čas računanja novih položajev delcev s pohitritvijo

število delcev	1.000	10.000	100.000	1.000.000
standardna funkcija sinus	0.3 ms	3.2 ms	35 ms	345 ms
vpogledna tabela funkcije sinus	0.3 ms	2.9 ms	33 ms	313 ms

6 Uporabimo vrednosti na območju [0, 2π)

6. Sklep

Naša animacija ognja deluje dovolj hitro za interaktiven izris več deset-tisočih delcev. V ustvarjenem orodju lahko premikamo in dodajamo izvore z različnim videzom in obnašanjem. Sistemu se lahko nastavi dovolj parametrov, tako bi ga lahko umestili v neko končno aplikacijo, na primer v 2D računalniško igro. Izvedba pa še ni dovolj realistična, da bi jo lahko uporabili v modernih 3D računalniških igrah.

Možnih je tudi nekaj izboljšav naše animacije. Lahko bi uporabili GPU za hitrejši izračun položajev delcev, vendar je ta del grafičnega cevovoda še precej nov in ni tako razširjen na večini osebnih računalnikov. Delce je možno upodobiti tudi na druge načine z različnimi metodami, kot npr. z risanjem prostorskih elementov (*volumetric rendering*), kar bo sicer upočasnilo izris. Za izboljšanje obnašanja delcev pa bi lahko uporabili kompleksnejšo dinamiko ognja, vendar pa bi se nam hitrost izvajanja pri tem zmanjšala.

Računalniška grafika se razvija hitro. Novejše tehnologije nam omogočajo vse več svobode pri programiranju grafičnih kartic. Z njihovo pomočjo bomo kmalu lahko že na vsaki zmogljivejši napravi hitreje upodabljali grafiko in sisteme delcev z zahtevnejšo dinamiko ter hitrejšim izrisom.

7. Seznam uporabljenih virov

- [Blinn, 1982]
  Blinn J. F. Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. *Computer Graphics*. 16, (1982), 3, str. 21-29.

- [Burg, 2014]
  Burg J. Building an Advanced Particle System. 2000. Dostopno na: http://www.gamasutra.com/view/feature/131565/building_an_advanced_particle_.php [1. 9. 2014].

- [Guid, 2001]
  Guid N. Računalniška grafika, Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko, 2001.

- [Lander, 2014]
  Lander J. The Ocean Spray in Your Face. 1998. Dostopno na: https://www.lri.fr/~mbl/ENS/IG2/devoir2/files/docs/particles.pdf [1. 9. 2014].

- [Reeves, 1983]
  Reeves W. T. Particle Systems – A technique for Modeling a Class of Fuzzy Objects. *ACM Transactions on Graphics*, 2, 1983, 2, str. 91-108.

- [Shiffman, 2012]
  Shiffman D. The Nature of Code: Simulating Natural Systems with Processing, Samozaložba, 2012.

- [Sims, 1990]
  Sims K. Particle Animation and Rendering Using Data Parallel Computation. *Computer Graphics*, 24, (1990), 4, str. 405-413.

- [Wikipedia, 2014]
  Particle system. Dostopno na: http://en.wikipedia.org/wiki/Particle_system [1. 9. 2014].

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

IZJAVA O AVTORSTVU

Spodaj podpisani/-a MARKO DOLAR

z vpisno številko E1021015

sem avtor/-ica diplomskega dela z naslovom:

ANIMACIJA OGNJA S SISTEMI DELCEV

(naslov diplomskega dela)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  RED. PROF. DR. NIKOLA GUID

  in somentorstvom (naziv, ime in priimek)

  VIŠ. PRED. DR. SIMON KOLMANIČ

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela.
- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne 25.9.2014

Podpis avtorja/-ice:

Dolar Marko

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

IZJAVA O USTREZNOSTI ZAKLJUČNEGA DELA

Podpisani mentor :

NIKOLA GUID

(ime in priimek mentorja)

in somentor (eden ali več, če obstajata):

SIMON KOLMANIČ

(ime in priimek somentorja)

Izjavljam (-va), da je študent

Ime in priimek: MARKO DOLAR

Št. indeksa: E1021015

Na programu: RAČUNALNIŠTVO IN INFORMACIJSKE TEHNOLOGIJE

izdelal zaključno delo z naslovom:

ANIMACIJA OGNJA S SISTEMI DELCEV

(naslov zaključnega dela v slovenskem in angleškem jeziku)

ANIMATION OF FIRE USING PARTICLE SYSTEMS

v skladu z odobreno temo zaključnega dela, Navodilih o pripravi zaključnih del in mojimi (najinimi oziroma našimi) navodili.

Preveril (-a, -i) in pregledal (-a, -i) sem (sva, smo) poročilo o plagiatorstvu.

Datum in kraj:
25. 9. 2014, Maribor

Podpis mentorja:
Nikola Guid

Datum in kraj:
Maribor, 25. 9. 2014

Podpis somentorja (če obstaja):
Simon Kolmanič

UNIVERZA V MARIBORU

Fakulteta za elektrotehniko, računalništvo in informatiko

IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUČNEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV

Ime in priimek diplomanta-tke: Marko Dolar

Vpisna številka: E1021015

Študijski program: FERI-E VS RAČUNALNIŠTVO IN INFORMACIJSKE TEHNOLOGIJE

Naslov diplomskega dela: ANIMACIJA OGNJA S SISTEMI DELCEV

Mentor: Nikola Guid

Somentor: Simon Kolmanič

Podpisani-a Marko Dolar izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Diplomsko delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija diplomskega dela je istovetna elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

Zaključno delo zaradi zagotavljanja konkurenčne prednosti, varstva industrijske lastnine ali tajnosti podatkov naročnika: _____
ne sme biti javno dostopno do _____ (datum odloga javne objave ne sme biti daljši kot 3 leta od zagovora dela).

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum diplomiranja, naslov diplomskega dela) na spletnih straneh in v publikacijah UM.

Datum in kraj: Maribor, 25.09.2014

Podpis diplomanta-tke: Dolar Marko

Podpis mentorja _____
(samo v primeru, če delo ne sme biti javno dostopno):

Podpis odgovorne osebe naročnika in žig: _____
(samo v primeru, če delo ne sme biti javno dostopno)